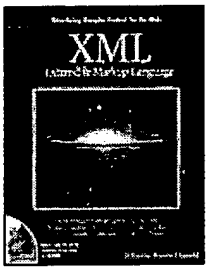
XML: Extensible Markup Language

Elliotte Rusty Harold



Our Price: $39.99
Readers' Advantage Price: $37.99 Join Now
In Stock: Ships within 24 hours

Same Day Delivery in Manhattan.

Format: Paperback, 1st ed., 426pp.
ISBN: 0764531999
Publisher: IDG Books Worldwide
Pub. Date: September 1998
Barnes & Noble Sales Rank: 625,077

Related Information

Bibliography
- Books by Elliotte Rusty Harold

About this Item
- From the Publisher
- Reviews
- Customer Reviews

From the Book
- Table of Contents
- Related Titles

From the Critics | Customer Reviews | Table of Contents

Write your own Review

2 other customers have reviewed this book. Average Rating:
Read what other customers have said

bn.com customers who bought this book also bought:

HTML & XHTML: The Definitive Guide, Chuck Musciano,Bill Kennedy
Cascading Style Sheets: The Definitive Guide, Eric A. Meyer,Richard Koman (Editor)
Dynamic HTML: The Definitive Reference, Danny Goodman,Paula Ferguson (Editor)
HTML and XML for Beginners: A Guide to Creating Web Pages, Michael Morrison
HTML 4 for Dummies with CD-ROM, Ed Tittel,Natanya Pitts,Chelsea Valentine

ABOUT THIS ITEM

From the Publisher

Not since Java has a new language turned so many heads in the Web community. Why is XML generating so much buzz? It offers greater flexibility and control when creating Web documents for one. If HTML doesn't have the tags you need, for example, make your own with XML. And that's just the beginning of what this powerful metamarkup language can do.

In *XML: Extensible Markup Language*, renowned author and programming guru Elliotte Rusty Harold combines clear, concise explanations with practical real-world examples to give you a complete understanding of XML. You get expert advice on creating XML documents, step-by-step instructions for adding customized structure to documents, tips for converting HTML to XML, strategies for assembling documents from multiple

books24x7

FEEDBACK

Search the Book Contents

Enter words or phrases that might occur in the text. Tips

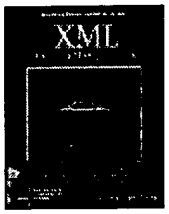
Search all books Search

Search only this Book

Look Up Books You Know

Enter title words, author's name, publisher or ISBN. Tips

Lookup



XML: Extensible Markup Language

by Elliotte Rusty Harold

ISBN: 076453

Hungry Minds © 1998, 426 pages

XML is here to stay, and this book gives you all the details how to use it to your best advantage.

Synopsis

This guide explains how to make the most of XML in developing Web pag and browser independent extensions. While this book is aimed at those wl know their way around web development, even those with little HTML or X background will find the information valuable. The author writes in a clear, accessible style that will help even novices get up to speed quickly. You'll a learn how to use XSL style sheets to convert documents into HTML, which handy if you've got legacy browsers.

Table of Contents

Back Cover

Deliver Rich Web Content

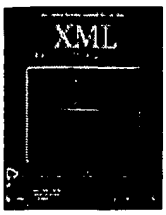
Make complex, structured content available on the Web regardless of bro or client software. With real-world examples, *XML* shows you how to use t fully internationalized language to create documents that are easy to trans across the Web and filled with readily reusable information -- from statistic and mathematics to graphics and multimedia.

Working Solutions for Web Development Challenges

- Expert strategies for creating XML documents and converting HTM XML
- Hands-on techniques for adding customized structure to document
- Detailed examples of using XSL style to design Web pages
- Practical solutions for assembling documents form multiple data sources
- Complete coverage of international scripts, character sets, fonts, a Unicode
- Advanced techniques for validating documents
- In-depth analysis of XLinks and XPointer
- Rules for writing well-formed XML documents

About the Author

Elliotte Rusty Harold is a New York City-based consultant and author. He Cafe con Leche, one of the most popular XML resource sites on the Web. is the author os IDG Books Woldwide's bestselling *Java Secrets* and *JavaBeans*.



Chapter 1 - Introducing XML

XML: Extensible Markup Language
Elliotte Rusty Harold

What Is XML?

XML stands for Extensible Markup Language (often misspelled as eXtensible Markup Language to justify the acronym). XML is a set of rules for forming semantic tags that break a document into parts and identify the different parts of the document.

XML Is a Metamarkup Language

XML isn't just another markup language like HTML or troff, which define a fixed set of tags describing a fixed number of elements. If your markup language of choice doesn't contain the tag you want or need, you're pretty much out of luck. You can wait for the next version of the markup language and hope that it includes the necessary tags, but you're really at the mercy of what vendors choose to include and support.

XML, however, is a metamarkup language, in which you make up the tags you need as you go along. These tags must be organized according to certain general principles, but they're quite flexible in their meaning. For instance, if you're working on genealogy and need to describe people, births, deaths, burial sites, families, marriages, divorces, and so forth, you can create tags for each of these elements. You don't have to try to force them to fit into paragraphs, list items, strong emphasis, or other very general categories.

The tags you create can be documented in a Document Type Definition (DTD). You learn more about DTDs in Part II of this book. For now, however, think of a DTD as a vocabulary and a syntax for certain kinds of documents. For example, Peter Murray-Rust's MOL.DTD describes a vocabulary and a syntax for the molecular sciences: chemistry, crystallography, solid state physics, and the like. This DTD can be shared and used by many different people in the field. Other DTDs are available for other disciplines, and you can create your own DTDs.

You may think that this approach is a nice idea in theory, but that in practice it's unlikely that Netscape and Internet Explorer will support thousands of different languages. The strength of XML, however, appears precisely when used as the template for these languages. XML defines a metasyntax that specific markup languages like CML, MathML, and others must follow. If an application understands this metasyntax, it automatically understands all languages built from this metalanguage.

A browser does not need to hard code each and every tag that may be used by thousands of different languages. Instead, the browser discovers the tags used by any given document as it reads the document or its DTD. The detailed instructions about how to render those tags are provided in a style sheet that's attached to the document.

For example, consider Schrodinger's equation:

$$i\hbar\frac{\partial\psi(r,t)}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2\psi(r,t)}{\partial x^2} + V(r)\psi(r,t)$$

Scientific papers are full of these equations, but scientists have been waiting eight years for browser vendors to support tags needed to write such equations. Musicians are in a similar fix—Netscape and Internet Explorer don't support sheet music.

XML means you don't have to wait for the browser vendors to catch up with your ideas. You can invent

the tags you need when you need them and tell the browsers how to display those tags.

XML Is a Semantic/Structured Markup Language

XML describes a document's structure and meaning. It does not describe the formatting of the elements on the page. Formatting can be added to documents through style sheets, but the document itself only contains tags that describe the contents of the document, not the appearance of the document.

By contrast, HTML encompasses both formatting, structural, and semantic tags. <B> is a formatting tag that makes its content bold. <STRONG> is a semantic tag that means its contents should be strongly emphasized. <TD> indicates that the contents are part of a table structure. In fact, some tags can have all three kinds of meaning. An <H1> header can mean the title of the page as well as "20 point Helvetica bold."

For example, a book in a list (you'll encounter this example in following chapters) could look like the following in HTML:

```
<dt>Java Secrets
<dd> by Elliotte Rusty Harold
<ul>
<li>Publisher: IDG Books Worldwide
<li>ISBN: 0-764-58007-8
<li>Pages: 900
<li>Price: $59.99
<li>Publication Date: May, 1997
<li>Bottom Line: Buy It
</ul><P>
The Java virtual machine, byte code, the sun packages, native
 methods, stand-alone applications, and a few more naughty bits.<P>
```

The book is described using a definition title, definition data, an unordered list, list items, and paragraphs. None of these elements has anything to do with a book. In XML, the same data could look as follows:

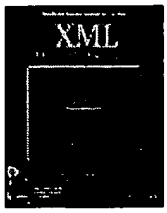
```
<book>
  <title>Java Secrets</title>
  <author>Elliotte Rusty Harold</author>
  <publisher>IDG Books Worldwide</publisher>
  <isbn>0-764-58007-8</isbn>
  <pages>900</pages>
  <price>59.99</price>
  <publication_date>May, 1997</publication_date>
  <recommendation>Buy It</recommendation>
  <blurb>
    The Java virtual machine, byte code, the sun packages, native
     methods, stand-alone applications, and a few more naughty bits.
  </blurb>
</book>
```

Instead of generic tags like <dt> and <li>, this listing uses meaningful tags like <book>, <title>, <author>, and <price>. This approach has a number of advantages, not the least of which is that it's somewhat easier for a human reading the source code to determine exactly what the author intended.

XML markup also makes it easier for nonhuman, automated robots to locate all the books in the document. In HTML, robots cannot ascertain more than that an element is a <dt>—they have no means of determining whether <dt> represents a book title, a definition, or a designer's favorite means of indenting text. Indeed, a single document may contain <dt> elements with all three meanings.

XML element names can be organized to have extra meaning in additional contexts. For example, they

may be the field names of a database. Not requiring a few fixed tags to serve all uses (as in HTML) makes XML far more flexible and amenable to varied uses.



Chapter 1 - Introducing XML

XML: Extensible Markup Language
Elliotte Rusty Harold

Why Is XML?

Why are people excited about XML? What can you do with XML that you can't easily do with existing technology? Because of the X in XML (eXtensible), there are many different reasons why people like it. Which ones interest you depends on your needs. But once you learn XML, you're likely to find it as the solution to more than one problem with which you're already struggling.

This section investigates some of the generic uses to which XML can be put. Later, you'll see some of the specific applications already developed with XML.

Domain Specific Markup Language

By allowing professions to develop their own common markup languages, XML enables individuals in the field to trade notes, data, and information without worrying about whether the person on the receiving end has the particular proprietary payware used to create the data. They can even send documents to people outside the profession with a reasonable confidence that they can at least view (even if not understand) the document.

This process can take place without creating bloatware or unnecessary complexity for those outside the profession. You may not be interested in a general way of describing electrical engineering diagrams, but electrical engineers are. You may not need to include sheet music in your Web pages, but composers do. XML lets the electrical engineers describe their circuits and the musicians notate their scores without (for the most part) stepping on each other's toes. Neither field needs special support from the browser manufacturers or complicated plug-ins as is now the case.

Common Data Format

A lot of computer data from the last forty years is irretrievably lost—not because of natural disaster or decaying backup media (though those are problems too, ones XML doesn't solve)—but simply because nobody bothered to document how one reads the data media and formats. A Lotus 1-2-3 file on a ten-year-old 5 1/4-inch floppy disk may be effectively irretrievable in most corporations today without a huge investment of time and resources. Data in a less-known binary format like Lotus Jazz may be gone forever.

At a low level, XML is an incredibly simple data format. XML can be written in 100 percent pure ASCII text as well as a few other well-defined formats. ASCII is reasonably resistant to corruption. Removing bytes or even large sequences of bytes does not noticeably corrupt the rest of the text. This starkly contrasts with many other formats, such as compressed data or serialized Java objects, where corruption or loss of even a single byte can render the entire remainder of the file unreadable.

Furthermore, XML is well-documented. The W3C's XML specification and numerous paper books (such as this one) tell you exactly how to read XML data. There are no secrets waiting to trip up the unwary.

At a higher level, XML is self-describing. Suppose you're an information archaeologist in the 23rd century and you encounter this chunk of XML code on an old floppy disk that has survived the ravages of time:

```
<person id="p1100" sex="M">
```

```
  <name>
    <given>Judson</given>
    <surname> McDaniel</surname>
  </name>
  <birth>
    <date>21 Feb 1834</date>  </birth>
  <death>
    <date>9 Dec 1905</date>  </death>
</person>
```

Even if you're not familiar with XML you can tell this data describes a man named Judson McDaniel who was born on February 21, 1834 and died on December 9, 1905. In fact, even with gaps in or corruption of the data, you could still probably extract most of this information. The same cannot be said for most proprietary spreadsheet or word processor formats.

Data Interchange

Because XML is easy to understand, nonproprietary, and easy to read and write, it's an excellent format for interchange of data between different applications. One such format under development is the Open Financial Exchange Format (OFX). OFX is designed to let personal finance programs like Microsoft Money and Quicken trade data. The data can be sent back and forth between programs and exchanged with banks, brokerage houses, and so forth.

XML is a nonproprietary format unencumbered by copyright, patent, trade secret, or any other sort of intellectual property restriction. While designed to be very expressive, XML is also easy for both human beings and computer programs to read and write. Thus, it's an obvious choice for these exchange languages.

XML helps users avoid getting locked into particular programs simply because their data is already written in such a program, or because a correspondent only accepts that program's proprietary format. In computer book publishing, most publishers require submissions in Microsoft Word. As a result, most computer book authors have to use Word, even if they would rather use WordPerfect or Nisus Writer. It's extremely difficult for other companies to gain market share unless they can read and write Word files. Reverse engineering the undocumented Word file format is a significant investment of limited time and resources. Most other word processors have a limited ability to read and write Word files, but generally lose track of styles, revision marks, and other important features. Because Word's document format is undocumented, proprietary, and constantly changing, Word tends to end up winning by default, even when writers would prefer to use other, simpler programs. If a common word processing format was developed in XML, writers could use their program of choice rather than the program required by their publisher.

Structured Data

XML is ideal for large and complex documents. It not only lets you specify a vocabulary for the document, but also lets you specify the relations between elements. For example, if you're putting together a Web page of sales contacts, you can require that every contact has a phone number and an e-mail address. If you're inputting data for a database, you can make sure that no fields are missing. You can require that every book have an author. You can even provide default values to be used when no data is entered.

XML also provides a client-side include mechanism that can integrate data from multiple sources and display it as a single document. The data can even be rearranged on the fly. Parts of the data can be shown or hidden depending on user actions. This is all extremely useful when you work with large information repositories like relational databases.

Chapter 1 - Introducing XML

XML: Extensible Markup Language
Elliotte Rusty Harold

Printer friendly format

Chapter 1 Contents

A Brief History of XML

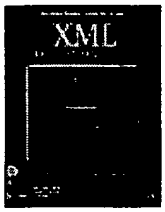
Like many standards, XML grew out of a lot of places around the same time. It's to point to one person or time and say, "This is where XML started." Nonetheles clearly has two main predecessors: SGML and HTML. Both were very successf markup languages, and HTML is probably the most widely used markup langua history. Nonetheless, both SGML and HTML have painfully obvious shortcomin new approach was needed to combine their strengths without their limitations—

The first impetus for XML was SGML, the Standard Generalized Markup Langu SGML is an international standard for semantic tagging of documents that's bee for a little over a decade. SGML is intended for semantic markup that assists cc cataloging and indexing, and can be extended in an infinite variety of ways to ha data formats.

SGML is popular in the defense sector and various other industries that need tc large quantities of highly structured data. SGML is unbelievably complex, howe mention expensive. For instance, the standard version of Adobe FrameMaker c about $850. Adobe FrameMaker+SGML starts at $1995 (I didn't omit a decimal point—that's one thousand, nine hundred, and ninety-five dollars), and that's on cheaper pieces of SGML software. SGML document management software rou breaks into tens of thousands of dollars or more. Although SGML has its advan is too complex and expensive for someone who wants to put their family tree or Internet.

The second pot of soil out of which XML grew was HTML. Invented by Tim Berr at CERN in 1990, HTML was designed as a simple replacement for SGML that comprehended by mere mortals and written without the assistance of expensive authoring tools. It succeeded beyond his wildest expectations.

Unfortunately, HTML didn



Chapter 1 - Introducing XML

XML: Extensible Markup Language
Elliotte Rusty Harold

A Brief History of XML

Like many standards, XML grew out of a lot of places around the same time. It's difficult to point to one person or time and say, "This is where XML started." Nonetheless, XML clearly has two main predecessors: SGML and HTML. Both were very successful markup languages, and HTML is probably the most widely used markup language in history. Nonetheless, both SGML and HTML have painfully obvious shortcomings, and a new approach was needed to combine their strengths without their limitations—XML.

The first impetus for XML was SGML, the Standard Generalized Markup Language. SGML is an international standard for semantic tagging of documents that's been around for a little over a decade. SGML is intended for semantic markup that assists computer cataloging and indexing, and can be extended in an infinite variety of ways to handle new data formats.

SGML is popular in the defense sector and various other industries that need to deal with large quantities of highly structured data. SGML is unbelievably complex, however, not to mention expensive. For instance, the standard version of Adobe FrameMaker costs about $850. Adobe FrameMaker+SGML starts at $1995 (I didn't omit a decimal point—that's one thousand, nine hundred, and ninety-five dollars), and that's one of the cheaper pieces of SGML software. SGML document management software routinely breaks into tens of thousands of dollars or more. Although SGML has its advantages, it is too complex and expensive for someone who wants to put their family tree on the Internet.

The second pot of soil out of which XML grew was HTML. Invented by Tim Berners-Lee at CERN in 1990, HTML was designed as a simple replacement for SGML that could be comprehended by mere mortals and written without the assistance of expensive authoring tools. It succeeded beyond his wildest expectations.

Unfortunately, HTML didn't scale very well. First of all, it was limited to a small fixed set of tags like <P>, <STRONG>, and <TITLE>. If the author needed a <BOOK>, <FATHER>, or <MOLECULE> tag, they were out of luck. HTML lacked the flexibility and adaptability of SGML.

In the early '90s, Marc Andreesen, Eric Bina, and a few others invented Mosaic at the University of Illinois, where they were working as cheap labor at NCSA for a few dollars an hour. Along the way, they added support for an <IMG> tag to HTML, and the Web took off. A couple of years later, Andreesen, Bina, and others, now at Netscape, began adding tags to Navigator at a furious rate, all of which became de facto standards with every new beta release. Throughout this process, however, no one worried about maintaining HTML's purity as an SGML application. In particular, semantic markup tags like <CODE>, <H1>, and <TABLE> were freely intermingled with stylistic markup tags like <CENTER>, <I>, and the notorious <BLINK>.

By 1996, the semantic roots of HTML had been thoroughly subverted by the needs of graphic designers. HTML tags no longer described the meaning of their content, but rather how the content would look when displayed by a Web browser. This approach posed significant challenges for indexing robots, disabled access, and many other uses. Interestingly these were all problems that SGML had solved years ago. At this point SGML user group meetings began to degenerate into long sour grapes sessions about how badly HTML sucked. Still SGML remained far too complicated and expensive for most uses.

In late 1994 and through 1995, Web sites began to grow. Old guard media companies like Time Warner and startups like c|net started to work on public Web sites on a previously unseen scale. Employees at large companies began putting more of their internal documents on private Web servers on the company LAN, simply because it was easier than distributing it on paper or in proprietary systems like Lotus Notes.

As these sites grew to hundreds of thousands of pages, many companies found themselves using large relational databases like Informix or Oracle to store Web content, which was then poured into templates using custom tags. Smaller sites began using products like Frontier or Cold Fusion to complete similar tasks. In essence, many different companies in different industries were growing into the realm where SGML-like solutions made sense, and they were all, more or less independently, reinventing their own version of SGML. In addition, these companies were paying tens of thousands of dollars or more to do this. Obviously, a new approach was needed.

Jon Bosak of Sun Microsystems started the W3C SGML working group (now called the XML working group) in the summer of 1996. The goal of the group was to create a sort of SGML lite that would bring SGML's strengths to the Web while still maintaining HTML's simplicity. Eventually, this goal would come to be called XML, though there were various arguments about the name. Other suggestions included MAGMA (Minimal Architecture for Generalized Markup Applications), SLIM (Structured Language for Internet Markup) and MGML (Minimal Generalized Markup Language)—in a few cases, the acronym seems to be more important than the full name.

XML is supposed to be a sort of SGML lite that retains the power and extensibility of SGML while being simple and cheap enough for mere mortals. To a large extent, this goal has been achieved. In regards to simplicity, the XML specification is an order of magnitude or more smaller than the SGML specification. In regard to power, there is little that can be done with SGML that can't be done with XML. In regard to cost, many free tools are already available for working with XML, with more anticipated soon. And Adobe has announced intentions to add XML support to FrameMaker—including the cheaper version.

Tim Bray was a prime mover in the development of XML and one of the two primary authors of the XML 1.0 specification. Bray had spent several happy years at "the best job in the world" managing the New Oxford English Dictionary Project. His experience with the OED informed his work on XML. In essence, he was trying to invent the markup language he wished he had back in 1987 when starting work on the OED. In particular, Bray wanted the following:

- a language simple enough for programmers to implement
- a language not limited to U.S. English
- documents easy for search engines to index

Tim Bray and C. M. Sperberg-McQueen of the University of Illinois wrote most of the original XML specification document. Many others contributed in one way or another.

A number of projects to produce markup languages for the Web for specialized domains occurred concurrently with the development of the official XML specification. The problems encountered by these languages heavily influenced XML's development. Peter Murray-Rust invented CML, the Chemical Markup Language, and the first general-purpose XML browser, Jumbo. At first, CML was supposed to be an SGML application, but it gradually transitioned to XML. Representatives of various companies involved in computer mathematics including Wolfram, IBM, Design Science, Waterloo Maple, the Geometry Center, and others contributed to the development of MathML for embedding mathematical equations in Web pages.

In November 1996, the first official draft specification for XML 1.0 was published, with numerous revisions over the next year.

April 1997 saw the first draft specification of XLL, the eXtensible Linking Language, primarily written by Eve Maler of ArborText and Steve DeRose of Inso Corp. and Brown University. Eve was also a major contributor to XML 1.0. XLL and its successors, XLink and XPointer, are discussed in Chapter 9. When fully implemented, XLL will allow more sophisticated and powerful linking than can be achieved with HTML and current browsers.

In July 1997, Microsoft released one of the first real applications of XML: the Channel Definition Format

(CDF) for pushing Web pages to subscribers. CDF is part of Internet Explorer 4.0 and is discussed further in Chapter 10.

Because XML is purely a structural and semantic format, some means is needed of describing the formatting of individual elements. One possibility is HTML's cascading style sheets. Another possibility, first introduced by Microsoft and Inso Corporation in August 1997, is XSL (eXtensible Style Language), which is introduced in Chapter 2 and explored thoroughly in Chapter 4.

In January 1998, Microsoft released the MSXSL processor, a free tool used extensively in this book for combining XML documents with XSL style sheets to produce HTML pages that can be viewed in standard browsers like Netscape or Internet Explorer 4.0. Future versions of these browsers should be able to view XML files directly without the need for an intermediate conversion step.

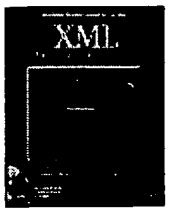
In February 1998, the W3C gave its official stamp of approval to Version 1.0 of the XML specification. This book is based upon Version 1.0. Other projects, including further development of XSL and XLL, are moving forward with new projects on the horizon.

An XML-Data proposal has been submitted to the W3C to provide for more structured data with which both object-oriented programmers and relational database aficionados are comfortable. The inclusion of a namespace mechanism that enables different DTDs to merge without conflicts is now a virtual certainty.

In late March 1998, Netscape shocked the XML world by not only releasing the source code for Mozilla 5.0 but also including more support for XML than expected. The XML community has already begun tinkering with the code, and exciting developments are sure to come by the time you're holding this book.

Third parties are busy as well, building the tools that enable new sorts of XML-based applications. Scripting gurus Dave Winer and Larry Wall are rearchitecting Frontier and Perl, respectively, to work with XML. Microsoft may make XML the native format for future versions of Word.

XML is gathering steam and is likely to take off as soon as the first stable version of Netscape 5.0 is released, which may be in your hands by the time you read this book. At the same time, a lot is already known about XML, and you can get a good seat on the XML train by boarding it now. As you'll see throughout this book, you can gain a lot by writing your Web pages in XML, even if you have to convert them to HTML before posting them.



Chapter 1 - Introducing XML

XML: Extensible Markup Language
Elliotte Rusty Harold

XML Programs

At the root, XML is a document format. It is a series of rules about what structures and text may appear in an XML document. The two levels of conformity to the XML standard are: well-formedness and validity. Part I of this book shows you how to write well-formed documents, while Part II shows you how to write valid documents. Part III shows you some applications of XML.

HTML is a document format designed for use on the Internet and inside Web browsers. XML can certainly be used for these purposes, as this book demonstrates. However, XML lends itself to far more general uses. It can be a storage format for word processors, a data interchange format for different programs, a means of enforcing conformity with intranet templates, and a way to preserve data in a human readable fashion.

Like all data formats, however, XML needs programs and content before it becomes useful. Thus, this book doesn't just discuss XML itself, which is little more than a specification for how data should be arranged. This book also discusses the programs you use to read and write data, and the content stored in XML. To understand XML, you also need to understand these.

XML documents are commonly created with an editor. This may be a basic text editor like Notepad or emacs that doesn't understand XML. It may be a WYSIWYG editor like Adobe FrameMaker that insulates you almost completely from the details of the underlying XML format. It may be something in between like JUMBO. For the most part, the fancy editors aren't useful yet, so this book concentrates on writing raw XML by hand in a text editor.

Other programs may also create XML documents. For example, Chapter 11 displays XML data straight from a FileMaker database. In this case, the data was first entered into the FileMaker database. Then, a FileMaker calculation field converted that data to XML. In general, XML works extremely well with databases.

In any case, the editor or other program creates an XML document. More often than not, this document is an actual file on some computer's hard disk, but this arrangement is not required. For example, the document may be a record or a field in a database or a stream of bytes received from a network.

An XML processor reads the document and verifies that the document's XML is well-formed. The processor may also check the validity of the document, though this test is not required. The exact details of these tests are covered in Chapter 3 and Chapters 5–8, respectively. Assuming the document passes the tests, the processor converts the document into a tree of elements.

Finally, the processor passes the tree or individual nodes of the tree to the end application. This application may be a browser like Netscape or another program that understands the data. A browser will display the data to the user. Other programs may also receive the data. For instance, the data may be interpreted as input to a database, a series of musical notes to play, or a Java program that should be launched. XML is extremely flexible and can be used for many different purposes.

All of these steps are independent and decoupled from each other. The XML document is the only connection between them. You can change the editor program independently of the end application. In fact, you may not always know what the end application is. It may be an end user reading your work, a database sucking in data, or an unknown future application. The data format is independent of the programs that read it.

HTML is also somewhat independent of the programs that read and write it, but it's really only suitable for browsing. Other uses, like database input, are outside its scope because HTML provides no way to force an author to include certain required content. For example, in HTML you have no way of requiring that every book has an ISBN number. In XML you can require this. You can even enforce the order in which particular elements appear. For example, you can require that `<H2>` headers must always follow `<H1>` headers.